Translation into any natural language of the error messages generated by any computer program

Bertrand Roehner[1]

Abstract Since the introduction of the Fortran programming language some 60 years ago, there has been little progress in making error messages more user-friendly. A first step in this direction is to translate them into the natural language of the students. In this paper we propose a simple script for Linux systems which gives word by word translations of error messages. It works for most programming languages and for all natural languages. Understanding the error messages generated by compilers is a major hurdle for students who are learning programming, particularly for non-native English speakers. Teachers and professors will squander a substantial amount of time translating the English vocabulary. By freeing them of this burden, an automatic translation will make more time available for the teaching of programming itself. Moreover, our personal experience showed us that many students simply do not have the patience to overcome this obstacle. Not only do they never become "fluent" in programming but many give up programming altogether. Whereas programming is a tool which can be useful in many human activities, e.g. history, genealogy, astronomy, entomology, in many countries the skill of programming remains confined to a narrow fringe of professional programmers. In all societies, besides professional violinists there are also amateurs. It should be the same for programming. Translating the terms used in error messages also gives an opportunity for explaining them. For instance, the words "operand" or "unary" are almost the same in English and in French, but they need to be explained because only few students know what they mean.
It is our hope that once translated and explained the error messages will be seen by the students as an aid rather than as an obstacle and that in this way more students will enjoy learning and practicing programming. They should see it as a funny game.

Version of 23 July 2015. Comments are welcome

Key-words: compiler, error messages, translation, programming, world languages.

1: Institute for Theoretical and High Energy Physics (LPTHE), Sorbonne Universités, University Pierre and Marie Curie (UPMC), Paris, France. Email: roehner@lpthe.jussieu.fr.

Motivation

Just in order to show that error messages are not always easy to understand, let me mention that while writing this paper I came across the following error message: “Unary operator expected”. Although hardly a beginner in programming, I had never before seen the word “unary”. Obviously, it was not a translation issue because the word is almost the same in English and in French, but rather a question of meaning. That is why it is important to say that our purpose is not only to translate but also to explain. In the present case the explanation is fairly simple: an unary operator expects only *one* argument.

The purpose of the present paper is to provide a procedure for translating (and explaining if necessary) error messages generated in English into the natural language Z preferred by the programmer. The English and Z versions will be displayed one under another in the command shell so that the Z version can provide an aid even if it is only a word by word translation.

Objections

When we started this project almost all the colleagues to whom we explained it raised objections and advised us to drop it. In this flow of objections the most common arguments were the following.

- All professional programmers, whatever their country of origin, know English. In other words, translating will be unnecessary and a waste of time.
- Understanding the meaning of some error messages may prove difficult even for native English speakers. This shows that it is not the natural language which is the main problem but rather the programming language.
- Error messages have been in English ever since the first programming languages were introduced in the early 1950s. This did not prevent the rapid development of computer science and the advent of the Internet Revolution.
- Anyway, said some colleagues, as the code is written in English, what is the point in translating only the error messages? Some colleagues asked us ironically: “May be, you also wish to rewrite the code into another language than English?”

As always in such discussions, these arguments were not wrong but they focused on specific aspects of the problem and discarded many others. Actually, at the root of it, there are two different conceptions of programming.

Narrow versus broad conception of programming

In the narrow conception, programming is reserved for professional people working for software companies on big projects. In this conception, programming will concern only a very small fraction of the population.

A parallel with the practice of music would be a society in which there are only professional musicians. In such a society there would be no musicians playing in a municipal band or orchestra, no persons singing in a choir, nobody playing music at home with friends just for personal satisfaction. Fortunately, except for a few Italian expressions (e.g. allegro vivace) there is no need to know another language before learning to read a score and play an instrument. A similar situation should prevail in programming.

In the broad conception a programming language is seen in a way not very different from a natural language that is to say as a means which allows people to interact with each other. Even a basic knowledge of Russian can make a visit to Moscow more enjoyable; similarly even just a working knowledge of a programming language will allow people to do things in their own field of interest which would be impossible otherwise. As an example, a knowledge of Java will allow historians to write a script which will search the Internet more effectively than through a manual search. In other words, instead of having to rely on standard search engines, they will be able to create their personal search engines focused on their own needs and requirements.

In the narrow conception the fact that error messages are in English is of little importance because through their education and professional training all programmers will soon get used to it.
In the broad conception the picture is different. If historians in Turkey, Japan or China want to use Java to create a personal search engine for a specific study, the fact that first of all they will have to learn (or to re-learn) English will certainly be a deterrent. It is true that the instructions of the Java code are themselves in English, but, as will be seen below, there is a great difference between code instructions and error messages.

Coping with error messages: a major hurdle for many students

On average over the past 50 years there have been very little changes in the formulation and presentation of error messages[1]. Fifty years ago programming was restricted to a tiny fraction of the population, mostly scientists for which the cryptic form of error messages was not a big problem. In 2015, programming is (or should be) used by a much broader fraction of the population. Teaching to audiences which comprised a fairly broad spectrum of the general public made us realize that for beginners coping with error messages was *the* major hurdle. Many of our students, especially those who did not have the opportunity to practice programming for days and weeks, were unable to overcome this obstacle and got discouraged. In short, in the present system occasional programmers are just left out in the cold.

[1] It is true that the introduction of IDE (Integrated Development Environment) tools has been a progress but it affected the error messages only indirectly.

As a confirmation of the key-role of the error messages, it can be observed that the few programming languages in which error messages have been made more user-friendly quickly become very popular among students. A case in point is the Python language.

Clearly, the fact that the error messages are in English is only a part of the problem. However, it adds an additional difficulty. To get a proper understanding of this point one should keep in mind two key-observations regarding English as a second language. They are explained in the following subsections.

English as a second language

Let us consider a French or Japanese tourist who visits London or Manhattan. In middle- and high-school he has had English lessons but in the meanwhile he forgot most of what he (or she) had learned. Thanks to his tourist guidebook he will be able to ask the appropriate question to a police officer:
“Excuse me Sir, I’m looking for the nearest post-office.” In response, he may get the following answer. “Well, you just have to walk across the park, the post-office is located just after the second block on your right. It may take you fifteen minutes at normal speed.”
Probably our poor tourist will understand barely one-half of the answer and most likely he will not be able to reach the post-office without asking the same question again to other persons.

What connection does this story have with programming? Very simple. Reading the question in the tourist guide is similar to writing programming code in English. It does not require a good knowledge of English because one needs to know only a limited number of pre-defined words.
On the contrary, the answer of the police officer is similar to the “answers” generated by the compiler in the sense that compilers use a broad set of words that are not pre-defined in any obvious way. In addition these words are included in sentences. Here is an example from a Fortran compiler:
“Invalid reference to variable in NAMELIST input”,

Actually, this example might well give a good reason for *not* doing any translation. Indeed, almost all the words used in this error message are exactly the same in English and in French. The only true English word may be “input” (entrée) but it is so commonly used in the pidgin form of French that is spoken nowadays that one can assume that it would be understood by almost everybody. As a confirmation, let us have a look at how Google translates the previous message.The translation reads as follows.
“Référence non valide à la variable en entrée NAMELIST”.

One can see that the translation is almost identical to the English version except for word order.

This leads us to our second point about English as a second language.

Scientific vocabulary in various languages

Between 2008 and 2014 one of us (in our research group) has had the opportunity to teach in Beijing and over those years he was fairly slow to realize that whereas almost all scientific words are the same in English and in French, in Chinese they are completely different. An obvious implication is that Chinese students, even those who are fairly fluent in English, will *not* understand the scientific vocabulary used in a course in physics or computer science unless they have devoted some special time and efforts to learn it. As shown in the table below, this observation about Chinese also applies to Hindi, Japanese or Korean.

Table 1 Examples of scientific words in various languages.

English	French	Spanish	Chinese	Hindi	Japanese	Korean	Russian
hydrogen	hydrogène	hidrógeno	qīng	hāidrōjana	suiso	su	vodorod
invalid	invalide	inválido	wúxiào	amānya	mukō	yuhyo	nevernyi
program	programme	programa	jìhuà	kāryakrama	puroguramu	peulogeulaem	programma
reference	référence	referencia	cānkǎo	sandarbha	rifarensu	chamgo	ssylka

Notes: In this short selection of words, Japanese is seen to be closer to English than is Chinese, Hindi or Korean. Incidentally, for a number of languages (e.g. Arabic, Hebrew, Vietnamese) Google-translation does not give phonetic transcriptions. That is why such languages were not included in the table.
Source: Google-translate (the word for "invalid" in Russian has been replaced by "nevernyi").

Which method?

How can one translate an error message generated by a compiler? We copy the message, paste it into the entry window of an online translator, copy the translation and finally paste it below the initial message. Thus, a first method would be to make this procedure automatic. However, this method relies on an online translator such as for instance Google-translate. Below, it will be explained why this method is not satisfactory.

We were told that an alternative method would be to use "gettext" together with "poedit". However, as we are not familiar with this software we did not try this option.

Here we will present another solution that we call the dictionary procedure in which

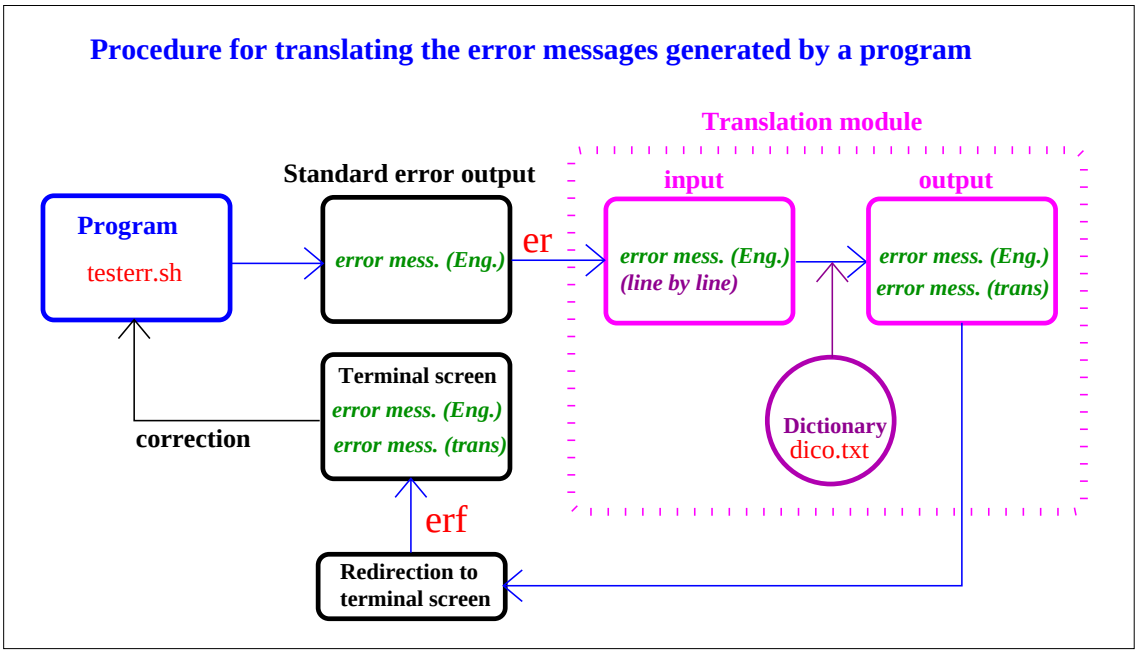

Fig. 1 Procedure for translating the error messages generated in the execution of a program. The words in red refer to the names of the files used in the script dicf.sh.

the translation module is simply an appropriate dictionary, whether self-made or web-based.

How Google-translate handles code words

The translations generated by Google-translation have both advantages and shortcomings.

Among the advantages is of course the fact that they are available for a broad range of languages. The second advantage is that, at least for Japanese and European languages (see Table 2 below) Google-translate more or less recognizes the words which belong to the code and does not attempt to translate them. "More or less" means that some code words nevertheless are translated. As shown in Table 2 for translations into French and German on average about 84% of the code words are not translated. As an illustration, one can mention the case of an "IF . . . THEN" instruction with respect to French and German. In the case of French, IF was not translated but THEN was translated into "Alors", whereas in German (for exactly the same error message) neither IF nor THEN were translated.

For Chinese, Hindi, Korean and Russian the percentage of the code words which were translated is much higher.

Instability of Google's translations

The fact (mentioned above) that translations into French and German behave differently is not very surprising. More puzzling is the fact that within one language the same sentence is translated differently when it occurs in separate error messages.

Table 2 Percentage of code words that were translated by Google-translate

	German	Japanese	French	Chinese	Korean	Russian	Hindi
Code words translated	9%	18%	23%	41%	50%	55%	68%

Notes: Ideally, none (i.e. 0%) of the words belonging to the computer code should be translated. Actually, the percentages range from 9% for German to 68% for Hindi. The fact that for 4 of the 7 major languages under consideration the percentage is higher than 40% suggests that in its present state Google-translate can hardly be used for translating error messages. The data are based on a sample of words such as NAMELIST, REWIND, OPEN, ENDFILE, VECTOR/INPUT and so on; in order to ensure context, they were submitted embedded within the messages in which they occurred.

Here is an illustration for translations into French. In one message “THE OFFENDING STATEMENT IS” is translated into the fairly funny sentence “LE COMPTE DE DÉLINQUANCE EST” whereas in another message the same words are correctly translated into “LA DÉCLARATION INCRIMINÉE EST”.

This instability creates a real problem because it makes almost impossible any attempt of introducing rules that would improve the translations provided by Google-translate.

Replacing the translation module by a dictionary

In Table 1 we observed that for many non-European languages it is the scientific vocabulary which is the major difficulty. Moreover, Table 2 showed that for these languages the translations generated by Google-translate were not very appropriate. Therefore, it may be tempting to replace the translation module by a self-made lexicon. Because it eliminates all syntactic difficulties such a dictionary is easy to create. For some programming languages (e.g. Fortran) lists of error messages are available on the Internet. Alternatively, at least for open source languages, the error messages may be extracted from the source code. In addition to translations, where required, the lexicon may also provide short explanations.

Although a lexicon may appear as a fairly rudimentary form of translation device, it may be quite useful nevertheless. Its initial implementation does not require much time and effort and, once implemented, it can be improved little by little by taking into account the observations and feedback of the students.

Appendix A: Code for automatic word by word translation

The code given below is independent of the language L into which one wishes to translate the error messages. For the purpose of illustration we will consider the case of L=French. The translations are contained in the dictionary represented by the file

`dico.txt` (see below) If one wishes to translate into another language, for instance Japanese, one must replace this file by an English to Japanese dictionary.

Regarding the programming language, we consider 4 cases: Bash, Python, Fortran and C. Because bash is just the language of Linux (or at least one of the versions) its execution will be the same on all Linux systems. For Python, Fortran and C there may be some slight differences from one Linux system to another.

Below we give the content of the dictionary, then the code of the scripts, then some explanations, and finally the results given by simple tests.

Dictionary: `dico.txt`

```
ENGLISH -> FRENCH DICTIONARY FOR ERROR MESSAGES
USED FOR MESSAGES GENERATED BY bash,python,C,Fortran
at=a l'endroit indique
before=avant
call=appel (par ex. appel a une instruction)
empty=vide
error=erreur
expected=attendu
file=fichier
forbidden=interdit
found=trouve
function=fonction
in=dans
last=en dernier
last)=en dernier
last):=en dernier lieu, a la fin
length=longueur
line=ligne
many=nombreux
matching=correspondant, ayant la meme forme
most=le plus
(most=le plus
near=pres de
number=nombre
operand=operande (3+5: 3 et 5 sont deux operandes)
statement=instruction, commande
token=occurrence d'un symbole
too=trop
traceback=remonter a la source
```

```
TypeError=type de l'erreur
unexpected=non attendu
unknown=inconnu
unsupported=non reconnu, incompatible avec le systeme
warning=avis (n'empeche pas l'execution)
without=sans
```

Two observations are in order.

- The translation can be supplemented with an explanation. This was done for the words: call, matching, operand, unsupported.
- The dictionary can be built little be little as the students are confronted with new error messages. Even if it is built by the professor before the start of the course, one should take advantage of the classes to make sure that the translations and explanations are well understood. If they are not, the dictionary may be improved.

Scripts used in the execution of a bash program

We need three kinds of scripts.

(1) A test-program called `testerr.sh` which contains an error and will therefore generate an error-message.

(2) The main script called `dicf.sh` which reads the error messages and produces the translations.

(3) The main script creates a new file called `erf`. It one runs the script a second time there will be a message saying "File erf already exists". In order to avoid this message, the file "erf" must be deleted before running dicf. This is done by the script: `frem.sh`

`testerr.sh`

```
#!/bin/bash
#BASH SCRIPT: PRODUCES OUTPUT (bonjour) AND ERROR (; ;)
#
 echo Bonjour
 echo a ; ;  echo b
```

`dicf.sh`

```
#!/bin/bash
# USAGE: bash dicf.sh .\testerr.sh
#  echo
#
# (1) EXECUTION + REDIRECTION OF ERROR MESSAGES TO err
#     ONE KEEPS ONLY THE FIRST n1 LINES OF err
```

```
#
 echo ’           RESULTATS’
 nl=10
 $1 2> err ;  head -$nl err > er
#
#     (2) ONE READS THE ERROR MESSAGES LINE BY LINE
#
 echo ’           ERREURS’
 li=0
 fd=dico.txt
 while read a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 ; do
 li=$(( $li+1 )) ; echo ; echo LIGNE $li
 echo $a1 $a2 $a3 $a5 $a6 $a7 $a8 $a9 $a10 $a11 $a12
#
#     (3) ONE TRANSLATES EACH LINE WORD BY WORD
#
 echo ’           TRADUCTION’ $li
 for i in $a1 $a2 $a3 $a5 $a6 $a7 $a8 $a9 $a10 $a11 $a12
 do
 grep -iw $i $fd
 done
 done < er
 echo
```

Explanations for `dicf.sh`

For the readers who are not familiar with the bash programming language the following explanations may be useful.

- A bash script f.sh can be executed either by `bash f.sh` or by `./f.sh`. In the second method the instruction has only one string. This string is given to `dicf.sh` as an argument.
- The key-instruction is: `$1 2> err`. `$1` reproduces the argument and therefore runs `testerr.sh`. The number “2” refers to the error output. Through `> err` this error output is redirected to the file `err` .
- Each of the variables `a1,a2,a3,...` reads one of the words in one line of the error message. It is the `while-do-done` loop which treats all the lines.
- For `i=$a1` the instruction `grep -iw $a1 $fd` searches the dictionary `$fd` for the word `$a1` and prints the line which contains the word along with its translation. The option `i` ignores case distinction, the option `w` selects only

matches that form whole words.

frem.sh

```
#!/bin/bash
# USAGE: ./frem f
# IF THE FILE f EXISTS IT WILL BE REMOVED
 f=$1
# pwd = print working directory
 p=$(pwd) ; fn=$p/$f
# THE ‘‘IF’’ IS A file test operator
 if [ -f $fn ] ; then rm -f $fn ; fi
```

How to run the scripts

In order to execute the test file `testerr.sh` and at the same time to translate the error messages one should type the following instruction on the command line:

```
bash frem.sh erf ; bash dicf.sh ./testerr.sh > erf
```

Output

One gets the following output contained in file `erf`.

erf

```
            RESULTATS
Bonjour
            ERREURS

LIGNE 1
./testerr.sh: line 6: error near unexpected token ‘;’
            TRADUCTION 1
line=ligne
error=erreur
near=pres de
unexpected=non attendu
token=occurrence d’un symbole

LIGNE 2
./testerr.sh: line 6: echo a ; ; echo b’
            TRADUCTION 2
line=ligne
```

Scripts used in the execution of programs in Python, Fortran and C

Nothing has to be changed in `dicf.sh`, only its argument needs to be replaced by the appropriate execution orders (see below).

Python

A possible test file containing an error is the following.

`testerr.py`

```
print("bonjour")
1 + "2"
```

This program will be executed by the instruction: `python testerr.py`.

It will generate the following error messages:

```
Traceback (most recent call last):
  File "testerr.py", line 2, in <module>
    1 + "2"
TypeError:
unsupported operand type(s) for +: 'int' and 'str'
```

In order to execute the python program and at the same time translate its error messages one should type the following instruction in the command line[2]:

```
bash frem.sh erf ; bash dicf.sh 'python testerr.py'>erf
```

Fortran

A possible test-file containing an error is the following[3]

`err.f90`

```
program tabulation
print *, "bonjour"
write *, ""
end program
```

This program will be executed by the instruction:

```
gfortran err.f90 -o errF
```

It will generate the following compilation error message:

```
err.f90:4.5:
    write *, ""
```

[2]For some reason the quotes around python testerr.py seem to look like counter-quotes, but they are in fact ordinary quotes '...'.

[3]For the cases of Fortran and C programs we consider the occurrence of an error in the compilation. This is for the sake of simplicity. In case the compilation is successful one needs to run the same procedure a second time with the argument of "dicf.sh" being "./errF".

```
      1
Error: Syntax error in WRITE statement at (1)
```

In order to execute the Fortran program and at the same time translate its error messages one should type the following instruction in the command line:

```
bash frem.sh erf ;
bash dicf.sh 'gfortran err.f90 -o errF'> erf
```

C

A possible test-file containing an error is the following.

err.c

```
#include <stdlib.h>
#include <stdio.h>

void main(){
    printf("bonjour\n");
    printf("\n")
}
```

This program will be executed by the instruction: `gcc err.c -o errC`

It will generate the following error message:

```
err.c: In function main:
    err.c:7:1: error: expected ; before } token
```

In order to execute the C program and at the same time translate its error message one should type the following instruction in the command line:

```
bash frem.sh erf ; bash dicf.sh 'gcc err.c -o errC'> erf
```

Appendix B: How Europe fell behind

It is hardly an exaggeration to say that all major programming languages in use nowadays have been created in the United States. A few decades ago some attempts in this direction had been made in Europe (e.g. Algol or Pascal) but they were quickly forgotten. It is interesting to understand why in this field European countries became less and less effective. To get a realistic assessment one must consider the different sectors of the software industry one by one: software for scientific computing, for the Internet, for graphics, images, films, translations, education, genealogy, and so forth and so on. In all these sectors the domination of US companies is overwhelming.

In this appendix we analyze the sociology of software development in order to find a clue for why Europe lags behind. It will be seen that a large part of the problem

comes from the fact that European countries have a narrow, purely professional, conception of “code writing” and software development. That conception is strongly connected with the fact that most European people, and particularly professors and students, see programming as a cryptic, purely technical activity. A more appropriate conception would be to see “code writing” as a language which allows people to create a whole new world, just as musical notation is the key for creation in the world of music.

The long process of software maturation

For most of the languages that we are talking about (Fortran, C, Java, Python), their development was a long process which covered several decades. Often it happened that the development of a software remained so to say in hibernation for years. That was the case for the “ImageMagick” software between 1990 and 1995 until new contributors took an interest in it. During such “sleeping beauty” episodes, instead of being dropped or frozen, the programs continued to be maintained until eventually brought back to life by a new team of interested people. Needless, to say this kind of development supposes a broad base of programmers, not only top programmers who work for major companies but also grass-root programmers. In addition, during this long maturation process developers and programmers needed to be supported by non-profit foundations or by government contracts.

The role played in the United States by non-profit foundations

As examples one can mention the “Free Software Foundation” and the “ImageMagick Studio Limited Liability Company”. Both are non-profit, tax-exempt organizations that were created some 30 years ago. Whereas the first one is a major actor in the free software sector, the second is a much smaller organization focused on image processing. Although in some cases (e.g. Unix or Python) the DARPA (Defense Advanced Research Projects Agency) was quite important, in a general way a myriad of foundations supported by donors and sponsors were the key-players.

The importance of software configuration management

Often contributions by various programmers to the projects harbored by such foundations are managed through a SCM (Software Configuration Management) software. The phase of integrating new code into an existing software is also referred to as “Software Verification and Validation”. It is obviously a crucial step because it allows software development to really become a collective creation even though the contributors do not have the opportunity to meet each other. For instance, as of May 2015, “GraphicsMagick” (a ramification of “ImageMagick”) was using a SCM called “Mercurial”. The website of “GraphicsMagick” also shows that over the past 10 years about 40 persons contributed by writing additional code; most of them did

so on an occasional part-time basis.

Connection between grass-roots and top programmers

Usually, good national football (soccer) teams exist in countries where football is popular and where each city has its own team of enthusiastic amateurs. In other words, the number and enthusiasm of grass-roots players conditions the achievements at the top. The same rule seems to apply to software development. In Europe, it seems that there is a very narrow base of programmers. From software engineers to developers, to programmers there are many possible definitions of the persons working in information technology. As a result it is hardly possible to find reliable comparative data[4]. However, it is common knowledge that in Europe business leaders as well as the European Commission routinely complain about a shortage of programmers.

Based on our teaching experience, it can be said that no more than 10% of the master students in physics are "fluent" in programming. True, most of them have a notion of several languages but only few practiced long enough to become really fluent. In fact, many students are discouraged at an early stage. The son of one of us was a case in point. During his second year as a master student in economics he got a course in programming. That was of course much too late in his education. In addition, confronted to a deluge of error messages and a lack of support from his teachers in this initial stage, he lost courage and decided that definitely programming was not his cup of tea.

Consequence of a lack of interaction between top and base: the case of CERN

Experiments in particle physics pose major challenges for computer scientists because one needs to record and sort out the information pertaining to billions of collisions between particles. This makes the CERN[5] a place where new software procedures are designed, implemented and used. Since the start of the LEP [6] project in 1983, computer scientists at CERN had to find new ways for making the experimental data available to teams of collaborators worldwide. This lead first to the development of the "N-upple" tool (1988) for sharing the data describing collision events and then, in 1989, to the introduction of the "World Wide Web" by Tim Berners-Lee et Robert Cailliau. The LHC (Large Hadron Collider) raised challenges of even greater magnitude because of a much larger number of collisions.

Yet, despite the breakthroughs made at CERN very little trickled down to the broader

[4]Typically, persons capable of writing (and debugging) a program of some 100 lines (in any language) may be defined as having a programmer capability. However, such a definition is difficult to implement at a statistical level.

[5]CERN is the acronym for "Conseil Européen pour la Recherche Nucléaire" (European Council for Nuclear Research). The organization was founded in 1954 by 12 European countries.

[6]Large Electron Positron collider, to date still the most powerful lepton collider ever built.

software community. Nowadays who knows that the World Wide Web started at CERN? As a matter of fact, 99% of the tools that allowed step-wise improvement of the Internet originated in the United States. This included for instance the development of Unix or of the software for creating, converting and transforming sounds, images and films. In other words, a trickling down process would have required a large base of persons in various fields capable of writing code and willing to cooperate with one another.

Even in the more narrow field of scientific research the diffusion of the tools created at CERN was neglected. For instance, the programming language called PAW (Physics Analysis at Workstations) which contained the innovative N-upple tool was frozen in 2004 and then integrated into a specialized computation software (called "root") almost only used by particle physicists. In other words, although this language was both powerful and pleasant to use no attempts were made to put it at the disposal of users outside of the community of particle physicists. Such a task would have been ideally suited for a foundation. Ensuring software maintenance and diffusion would have been a way to fully exploit the work and efforts of the developers of PAW over more than 15 years.

Conclusions

To conclude this short inquiry into the sociology of software development one can say three things.

- Rather than a top-down process, software development should rather be seen as a bottom-up growth process. In other words, the achievements of top developers depend upon the interest, capability and activity of a broad base.
- There is good reason to think that an initial unfriendly contact with programming languages discourages many students, thus preventing the constitution of a broad base of programmers especially in countries whose natural languages have minimal overlap with English.
- It is true that a number of gifted persons from various countries, like Vinod Khosla from India and founder of "Sun Microsystems", Guido van Rossum from the Netherlands and creator of Python, Linus Torvalds from Finland and a main developer of Linux, were able to make major contributions, but it is also true that all three moved to California fairly early in their carrier.

Acknowledgments The author wishes to express his gratitude to his colleagues Marc Bellon, Vladimir Dotsenko, Harold Erbin, Marco Picco and Hugo Ricateau for their help and cheerfulness. He is also grateful to Gilles Dowek (INRIA), Christophe Gragnic and Hiroshi Iyetomi (University of Niigata) for their interest in this project and their advice.